Using Vault for Secret Management: A Practical Guide with Kubernetes

Project Overview

This project demonstrates how to use **HashiCorp Vault** for secure secret management within Kubernetes. We'll explore real-world use cases for securely storing and retrieving secrets such as:

1. API URLs Done by **Mohamad**
2. Username and password Done by **Ammar**
3. Token-Based API Access Done by **Ali**
4. Database connection strings Done by **Yehia**

Vault will be integrated with **Kubernetes**.

Contributors

- Mohamad *(API Fetcher use case)*
- Ali *(Token-Based API use case)*
- Yehia *(Database URL connection use case)*
- Ammar *(Username/password login use case)*

Vault Architecture Overview

HashiCorp Vault is designed to securely store, manage, and control access to secrets (API keys, passwords, certificates) through a centralized, encrypted system. Its architecture consists of:

1. Core Components

- **Storage Backend** – Persists encrypted secrets (supports Consul, DynamoDB, PostgreSQL, etc.).
- **Secret Engines** – Plugins that handle secrets (KV, PKI, AWS, databases).
- **Auth Methods** – Controls user/application access (LDAP, JWT, Kubernetes, OIDC).
- **API/CLI** – Interfaces for interactions (RESTful API and vault CLI).

2. Security Layers

- **Encryption** – All data is encrypted before storage (AES-256-GCM).
- **Lease/TTL** – Secrets are dynamically issued and automatically revoked.
- **Audit Logging** – Tracks all requests for compliance.

3. High Availability (HA)

- Supports multi-server clusters with leader-follower replication.
- Auto-failover ensures uptime if a node fails.

4. Extensibility

- Plugins for custom secret engines, auth methods, and storage backends.
- Vault's architecture ensures zero-trust security, scalability, and seamless integration with cloud/on-prem systems.

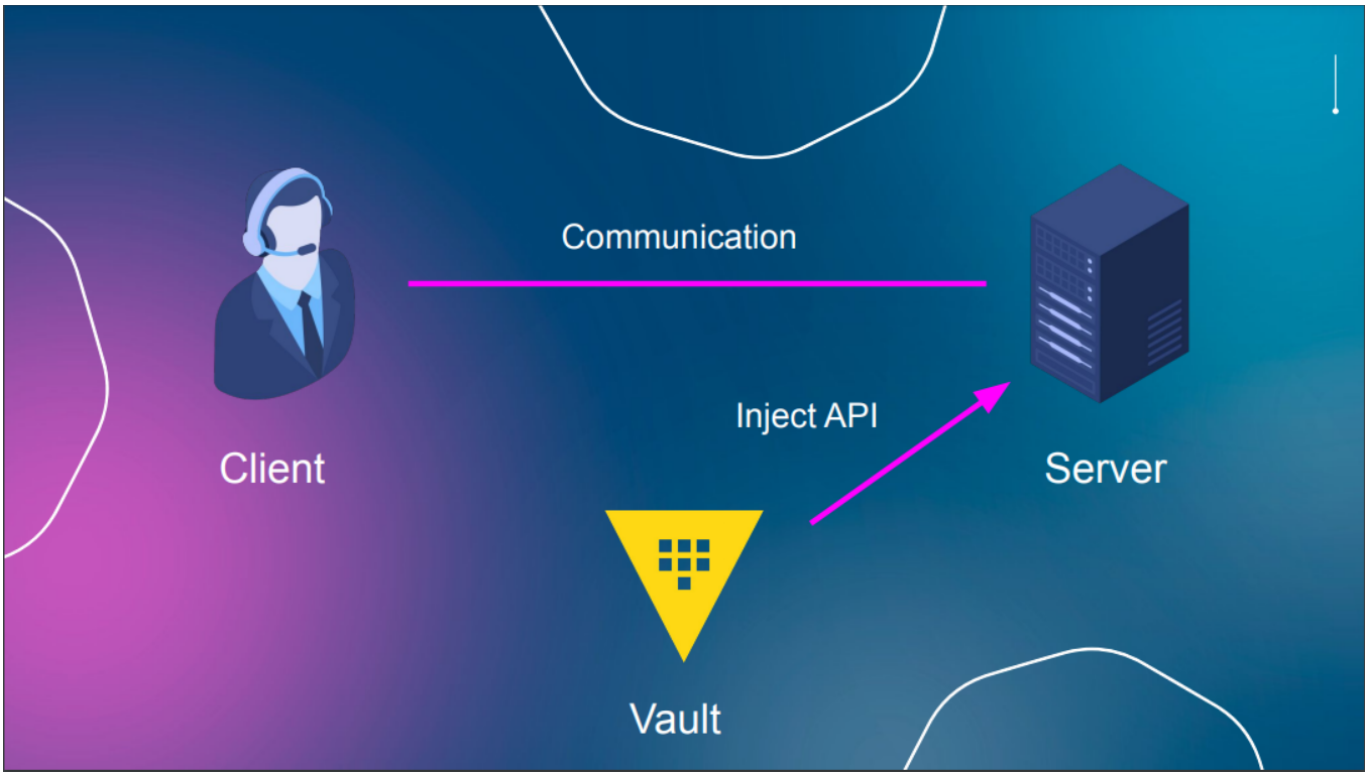
Setup

- 2 AWS EC2 instances were initialized for this project choosing `ubuntu` as operating system
- One instance was used for the (API + Token) cases
- The other instance was used for the (Login + Database) cases
- For both instances the following steps were followed at the beginning:
  - Installing `kubectl` following this guide
  - Installing `minikube` following this guilde and using `kubectl` as an alias for `minikube kubectl --`
  - Installing `docker` and adding it to `sudo group` following this guide
  - Installing `helm` following this guide and activating `vault` inside it following this guide

---

Use Case 1: API Fetcher App with Vault (By Mohamad)

In this section, we will deploy a Kubernetes-based application that securely fetches API data using secrets stored in HashiCorp Vault.



1. Verify Vault

Confirm the pod is running:

```
kubectl get pods
```

```
Last login: Mon Apr 21 12:11:14 2025 from 45.67.35.120
ubuntu@ip-172-31-31-31:~$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
vault-0                                 1/1     Running   0          77m
vault-agent-injector-75f9dfc9c8-gnmp9   1/1     Running   0          77m
ubuntu@ip-172-31-31-31:~$
```

2. Configure Vault Secrets

Enable the KV v2 secrets engine and store your API URL secret:

```
# Connect to the Vault pod shell
kubectl exec -it vault-0 -- /bin/sh

# Enable KV v2 secrets engine at custom path "internal"
vault secrets enable -path=internal kv-v2

# Add the API URL secret to Vault
vault kv put internal/database/config
api_url="https://jsonplaceholder.typicode.com/posts"

# Retrieve the secret to verify it was added
vault kv get internal/database/config
```

```
ubuntu@ip-172-31-31-31:~$ kubectl exec -it vault-0 -- /bin/sh
/ $ vault secrets enable -path=internal kv-v2
Success! Enabled the kv-v2 secrets engine at: internal/
/ $ vault kv put internal/database/config api-url="https://jsonplaceholder.typicode.com/posts"
======== Secret Path ========
internal/data/database/config

======= Metadata =======
Key                Value
---                -----
created_time       2025-04-23T16:56:04.78719343Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            1
/ $ vault kv get internal/database/config
======== Secret Path ========
internal/data/database/config

======= Metadata =======
Key                Value
---                -----
created_time       2025-04-23T16:56:04.78719343Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            1

===== Data =====
Key        Value
---        -----
api-url    https://jsonplaceholder.typicode.com/posts
```

3. Configure Kubernetes Authentication

Inside the Vault pod:

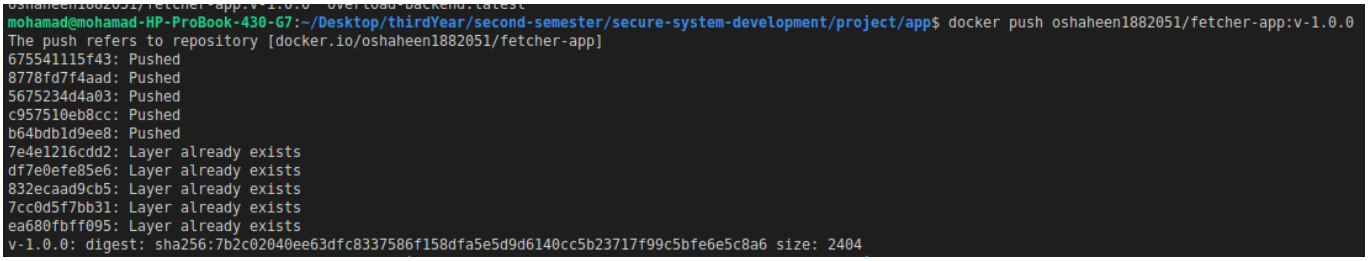
```
# Enable the Kubernetes authentication method
vault auth enable kubernetes

# Configure Vault with the Kubernetes cluster API endpoint
vault write auth/kubernetes/config \
    kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"

# Define a policy to allow read access to the API URL secret
vault policy write api-fetcher - <<EOF
path "internal/data/database/config" {
  capabilities = ["read"]
}
EOF

# Create a Kubernetes authentication role for the app
vault write auth/kubernetes/role/api-fetcher \
    bound_service_account_names=api-fetcher \
    bound_service_account_namespaces=default \
    policies=api-fetcher \
    ttl=24h

# Exit the Vault container
exit
```

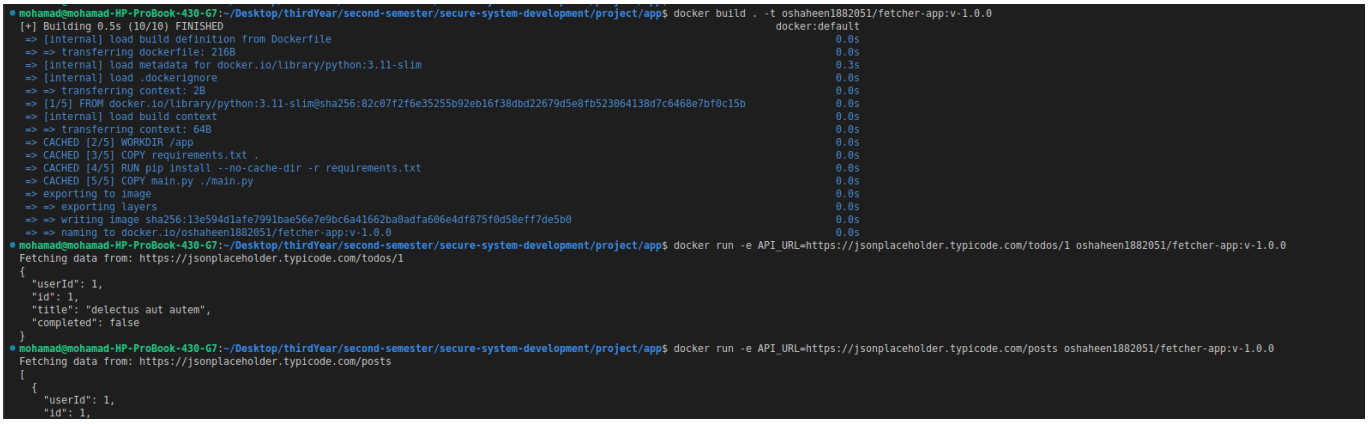
```
ubuntu@ip-172-31-31-31:~$ kubectl exec -it vault-0 -- /bin/sh
/ $ vault auth enable kubernetes

Success! Enabled kubernetes auth method at: kubernetes/
/ $
/ $ vault write auth/kubernetes/config \
>     kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"
Success! Data written to: auth/kubernetes/config
/ $ vault policy write api-fetcher - <<EOF
> path "internal/data/database/config" {
>   capabilities = ["read"]
> }
> EOF
Success! Uploaded policy: api-fetcher
/ $ vault write auth/kubernetes/role/api-fetcher \
>     bound_service_account_names=api-fetcher \
>     bound_service_account_namespaces=default \
>     policies=api-fetcher \
>     ttl=24h
Success! Data written to: auth/kubernetes/role/api-fetcher
/ $ exit
ubuntu@ip-172-31-31-31:~$
```

4. Build the API Fetcher App

This Flask-based app fetches data from an external API defined in the `API_URL` environment variable or query parameter.

```
# Build the Docker image
docker build . -t oshaheen1882051/fetcher-app:v-1.0.0 --no-cache

# Push the image to Docker Hub
docker push oshaheen1882051/fetcher-app:v-1.0.0
```





Docker Hub Link for fetcher-app

5. Create Helm Chart for `api-fetcher`

Create the chart:

```
helm create api-fetcher
```

Update `values.yaml`

```
image:
  repository: oshaheen1882051/fetcher-app
  pullPolicy: Always
  tag: "v-1.0.0"

serviceAccount:
  create: true
  automount: true
  name: "api-fetcher"

vault:
  secretPath: "internal/data/database/config"
  secretKey: "api_url"
```

```
    role: "api-fetcher"

  service:
    type: ClusterIP
    port: 8080
```

Update `templates/deployment.yaml` annotations

```
  annotations:
    vault.hashicorp.com/agent-inject: "true"
    vault.hashicorp.com/role: "{{ .Values.vault.role }}"
    vault.hashicorp.com/agent-inject-secret-database-config.txt: "{{
.Values.vault.secretPath }}"
    vault.hashicorp.com/agent-inject-template-database-config.txt: |
      {{`{{- with secret "`}}{{ .Values.vault.secretPath }}{{`" -}}`}}
      API_URL={{`{{ .Data.data.api_url }}`}}
      {{`{{- end }}`}}
```

6. Deploy the Chart

```
# Install the Helm chart for the application
helm install api-fetcher ./api-fetcher/
```

```
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=api-fetcher,app.kubernetes.io/instance=api-fetcher" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
ubuntu@ip-172-31-31-31:~$ kubectl get po
NAME                                    READY   STATUS    RESTARTS      AGE
api-fetcher-f86b49797-6625p             2/2     Running   0             5s
vault-0                                 1/1     Running   3 (47m ago)   9h
vault-agent-injector-75f9dfc9c8-w9f7g   1/1     Running   3 (56m ago)   9h
ubuntu@ip-172-31-31-31:~$ kubectl exec -it api-fetcher-f86b49797-6625p -- /bin/bash
Defaulted container "api-fetcher" out of: api-fetcher, vault-agent, vault-agent-init (init)
root@api-fetcher-f86b49797-6625p:/app# cat /vault/secrets/database-config.txt
API_URL=https://jsonplaceholder.typicode.com/posts
```

7. Test the API Fetcher App

Forward the service port and make a request:

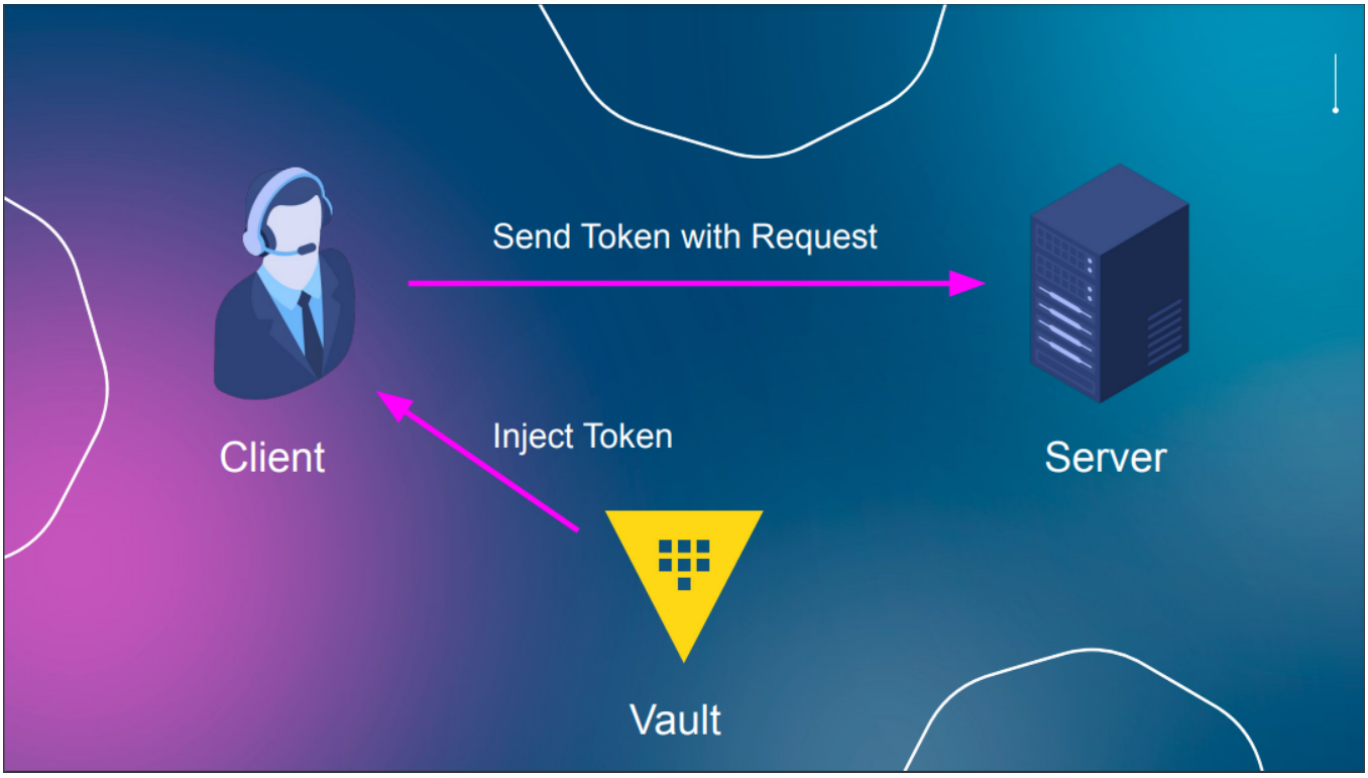
```
# Forward port 8080 from the pod to localhost
kubectl port-forward api-fetcher-f86b49797-6625p 8080:8080

# Send a test request to the running app
curl http://localhost:8080
```

```
exit
ubuntu@ip-172-31-31-31:~$ kubectl port-forward api-fetcher-f86b49797-6625p 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
```

```
ubuntu@ip-172-31-31-31:~$ curl http://localhost:8080
[{"body":"quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto","id":1,"title":"sunt aut facere repell
at provident occaecati excepturi optio reprehenderit","userId":1},{"body":"est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil moles
tiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla","id":2,"title":"qui est esse","userId":1},{"body":"et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloreib
us vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut","id":3,"title":"ea molestias quasi exercitationem repellat qui ipsa sit aut","userId":1},{"body":"ullam et saepe reiciendis vol
uptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit","id":4,"title":"eum et est occaecati","userId
":1},{"body":"repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolor neque","id":5,"title":"nesciunt quas odio","userId":
1},{"body":"ut aspernatur corporis harum nihil quis provident sequi\nmollitia nobis aliquid molestiae\nperspiciatis et ea nemo ab reprehenderit accusantium quas\nvoluptate dolores velit et doloremque molestiae",
"id":6,"title":"dolorem eum magni eos aperiam quia","userId":1},{"body":"dolore placeat quibusdam ea quo vitae\nmagni quis enim qui quis quo nemo aut saepe\nquidem repellat excepturi ut quia\nsunt ut sequi eos e
a sed quas","id":7,"title":"magnam facilis autem","userId":1},{"body":"dignissimos aperiam dolorem qui eum\nfacilis quibusdam animi sint suscipit qui sint possimus cum\nquaerat magni maiores excepturi\nipsam ut
commodi dolor voluptatum modi aut vitae","id":8,"title":"dolorem dolore est ipsam","userId":1},{"body":"consectetur animi nesciunt iure dolore\nenim quia ad\nveniam autem ut quam aut nobis\net est aut quod aut p
rovident voluptas autem voluptas","id":9,"title":"nesciunt iure omnis dolorem tempora et accusantium","userId":1},{"body":"quo et expedita modi cum officia vel magni\ndoloribus qui repudiandae\nvero nisi sit\nqu
os veniam quod sed accusamus veritatis error","id":10,"title":"optio molestias id quia eum","userId":1},{"body":"delectus reiciendis molestiae occaecati non minima eveniet qui voluptatibus\naccusamus in eum beat
ae sit\nvel qui neque voluptates ut commodi qui incidunt\nut animi commodi","id":11,"title":"et ea vero quia laudantium autem","userId":2},{"body":"itaque id aut magnam\npraesentium quia et ea odit et ea volupta
s et\nsapiente quia nihil amet occaecati quia id voluptatem\nincidunt ea est distinctio odio","id":12,"title":"in quibusdam tempore odit est dolorem","userId":2},{"body":"aut dicta possimus sint mollitia volupta
s commodi quo doloremque\niste corrupti reiciendis voluptatem eius rerum\nsit cumque quod eligendi laborum minima\nperferendis recusandae assumenda consectetur porro architecto ipsum ipsam","id":13,"title":"dolo
rum ut in voluptas mollitia et saepe quo animi","userId":2},{"body":"fuga et accusamus dolorum perferendis illo voluptas\nnon doloremque neque facere\nad qui dolorum molestiae beatae\nsed aut voluptas totam sit
illum","id":14,"title":"voluptatem eligendi optio","userId":2},{"body":"reprehenderit quos placeat\nvelit minima officia dolores impedit repudiandae molestiae nam\nvoluptas recusandae quis delectus\nofficiis har
um fugiat vitae","id":15,"title":"eveniet quod temporibus","userId":2},{"body":"suscipit nam nisi quo aperiam aut\nasperiores eos fugit maiores voluptatibus quia\nvoluptatem quis ullam qui in alias quia est\ncon
sequatur magni mollitia accusamus ea nisi voluptate dicta","id":16,"title":"sint suscipit perspiciatis velit dolorum rerum ipsa laboriosam odio","userId":2},{"body":"eos voluptas et aut odit natus earum\nasperna
tur fuga molestiae ullam\ndeserunt ratione qui eos\nqui nihil ratione nemo velit ut aut id quo","id":17,"title":"fugit voluptas sed molestias voluptatem provident","userId":2},{"body":"eveniet quo quis\nlaborum
totam consequatur non dolor\nut et est repudiandae\nest voluptatem vel debitis et magnam","id":18,"title":"voluptate et itaque vero tempora molestiae","userId":2},{"body":"illum quis cupiditate provident sit mag
nam\nea sed aut omnis\nveniam maiores ullam consequatur atque\nadipisci quo iste expedita sit quos voluptas","id":19,"title":"adipisci placeat illum aut reiciendis qui","userId":2},{"body":"qui consequuntur duci
mus possimus quisquam amet similique\nsuscipit porro ipsam amet\neos veritatis officiis exercitationem vel fugit aut necessitatibus totam\nomnis rerum consequatur expedita quidem cumque explicabo","id":20,"title
":"doloribus ad provident suscipit at","userId":2},{"body":"repellat aliquid praesentium dolorem quo\nsed totam minus non itaque\nnihil labore molestiae sunt dolor eveniet hic recusandae veniam\ntempora et tenet
ur expedita sunt","id":21,"title":"asperiores ea ipsam voluptatibus modi minima quia sint","userId":3},{"body":"eos qui et ipsum ipsam suscipit aut\nsed omnis non odio\nexpedita earum mollitia molestiae aut atqu
e rem suscipit\nnam impedit esse","id":22,"title":"dolor sint quo a velit explicabo quia nam","userId":3},{"body":"veritatis unde neque eligendi\nquae quod architecto quo neque vitae\nest illo sit tempora dolore
nque fugit quod\net et vel beatae sequi ullam sed tenetur perspiciatis","id":23,"title":"maxime id vitae nihil numquam","userId":3},{"body":"enim et ex nulla\nomnis voluptas quia qui\nvoluptatem consequatur numq
uam aliquam sunt\ntotam recusandae id dignissimos aut sed asperiores deserunt","id":24,"title":"autem hic labore sunt dolores incidunt","userId":3},{"body":"ullam consequatur ut\nomnis quis sit vel consequuntur\
nipsa eligendi ipsum molestiae et omnis error nostrum\nmolestiae illo tempore quia et distinctio","id":25,"title":"rem alias distinctio quo quis","userId":3},{"body":"similique esse doloribus nihil accusamus\nom
nis dolorem fuga consequuntur reprehenderit fugit recusandae temporibus\nperspiciatis cum ut laudantium\nomnis aut molestiae vel vero","id":26,"title":"est et quae odit qui non","userId":3},{"body":"eum sed dolo
res ipsam sint possimus debitis occaecati\ndebitis qui qui et\nut placeat enim earum aut odit facilis\nconsequatur suscipit necessitatibus rerum sed inventore temporibus consequatur","id":27,"title":"quasi id et
 eos tenetur aut quo autem","userId":3},{"body":"non et quaerat ex quae ad maiores\nmaiores recusandae totam aut blanditiis mollitia quas illo\nut voluptatibus voluptatem\nsimilique nostrum eum","id":28,"title":
"delectus ullam et corporis nulla voluptas sequi","userId":3},{"body":"odit magnam ut saepe sed non qui\ntempora atque nihil\naccusamus illum doloribus illo dolor\neligendi repudiandae odit magni similique sed c
um maiores","id":29,"title":"iusto eius quod necessitatibus culpa ea","userId":3},{"body":"alias dolor cumque\nimpedit blanditiis non eveniet odio maxime\nblanditiis amet eius quis tempora quia autem rem\na prov
ident perspiciatis quia","id":30,"title":"a quo magni similique perferendis","userId":3},{"body":"debitis eius sed quibusdam non quis consectetur vitae\nimpedit ut qui consequatur sed aut in\nquidem sit nostrum
et maiores adipisci atque\nquaerat voluptatem adipisci repudiandae","id":31,"title":"ullam ut quidem id aut vel consequuntur","userId":4},{"body":"deserunt eos nobis asperiores et hic\nest debitis repellat moles
tiae optio\nnihil ratione ut eos beatae quibusdam distinctio maiores\nearum voluptates et aut adipisci ea maiores voluptas maxime","id":32,"title":"doloremque illum aliquid sunt","userId":4},{"body":"rerum ut et
 numquam laborum odit est sit\nid qui sint in\nquasi tenetur tempore aperiam et quaerat qui in\nrerum officiis sequi cumque quod","id":33,"title":"qui explicabo molestiae dolorem","userId":4},{"body":"ea velit p
erferendis earum ut voluptatem voluptate itaque iusto\ntotam pariatur in\nnemo voluptatem voluptatem autem magni tempora minima in\nest distinctio qui assumenda accusamus dignissimos officia nesciunt nobis","id"
:34,"title":"magnam ut rerum iure","userId":4},{"body":"nisi error delectus possimus ut eligendi vitae\nplaceat eos harum cupiditate facilis reprehenderit voluptatem beatae\nmodi ducimus quo illum voluptas elige
ndi\net nobis quia fugit","id":35,"title":"id nihil consequatur molestias animi provident","userId":4},{"body":"ad mollitia et omnis minus architecto odit\nvoluptas doloremque maxime aut non ipsa qui alias venia
m\nblanditiis culpa aut quia nihil cumque facere et occaecati\nqui aspernatur quia eaque ut aperiam inventore","id":36,"title":"fuga nam accusamus voluptas reiciendis itaque","userId":4},{"body":"debitis et eaqu
e non officia sed nesciunt pariatur vel\nvoluptatem iste vero et ea\nnumquam aut expedita ipsum nulla in\nvoluptates omnis consequatur aut enim officiis in quam qui","id":37,"title":"provident vel ut sit ratione
 est","userId":4},{"body":"animi esse sit aut sit nesciunt assumenda eum voluptas\nquia voluptatibus provident quia necessitatibus ea\nrerum repudiandae quia voluptatem delectus fugit aut id quia\nratione optio
eos iusto veniam iure","id":38,"title":"explicabo et eos deleniti nostrum ab id repellendus","userId":4},{"body":"corporis rerum ducimus vel eum accusantium\nmaxime aspernatur a porro possimus iste omnis\nest in
 deleniti asperiores fuga aut\nvoluptas sapiente vel dolore minus voluptatem incidunt ex","id":39,"title":"eos dolorem iste accusantium est eaque quam","userId":4},{"body":"ut voluptatum aliquid illo tenetur nem
```

Use Case 2: Token-Based API Access (By Ali)

In this section we will show how to store an API URL and access token in Vault, inject them into a Kubernetes pod, and use them in a Flask app to authenticate requests.



1. Generate and Store the Token in Vault

a. Generate a random token locally

```
# Generate a 128-bit (16-byte) hex token
echo "$(openssl rand -hex 16)"
# Example output: 0e5ca54075b7af0e785b75fe0f708517
```

b. Store the token and API URL in Vault

```
vault kv put internal/token/api-client \
  api_url="https://jsonplaceholder.typicode.com/posts" \
  token="0e5ca54075b7af0e785b75fe0f708517"
```

2. Configure Vault Policy and Kubernetes Role

1. **Create a Vault policy** that grants read access to the token path:

   ```
   vault policy write extended-api-fetcher - <<EOF
   path "internal/data/token/api-client" {
     capabilities = ["read"]
   }
   EOF
   ```

2. **Define a Kubernetes auth role** using that policy:

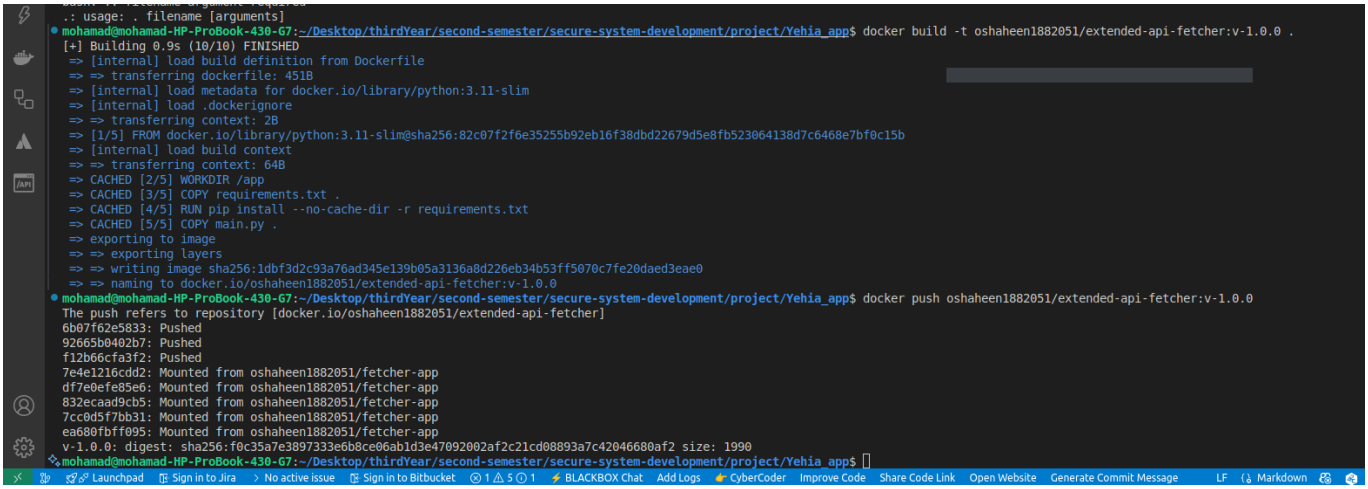
   ```
   vault write auth/kubernetes/role/extended-api-fetcher \
     bound_service_account_names=extended-api-fetcher \
     bound_service_account_namespaces=default \
     policies=extended-api-fetcher \
     ttl=24h
   ```

```
ubuntu@ip-172-31-31-31:~$ kubectl exec -it vault-0 -- /bin/sh
/ $ vault secrets enable -path=internal kv-v2
Success! Enabled the kv-v2 secrets engine at: internal/
/ $ vault kv put internal/token/api-client \
>   api_url="https://jsonplaceholder.typicode.com/posts" \
>   token="0e5ca54075b7af0e785b75fe0f708517"
========= Secret Path =========
internal/data/token/api-client

======= Metadata =======
Key                Value
---                -----
created_time       2025-04-25T09:22:06.234407424Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            1
/ $ vault kv get internal/database/config
No value found at internal/data/database/config
/ $ vault kv get internal/token/api-client
========= Secret Path =========
internal/data/token/api-client

======= Metadata =======
Key                Value
---                -----
created_time       2025-04-25T09:22:06.234407424Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            1

===== Data =====
Key        Value
---        -----
api_url    https://jsonplaceholder.typicode.com/posts
token      0e5ca54075b7af0e785b75fe0f708517
/ $ vault auth enable kubernetes

Success! Enabled kubernetes auth method at: kubernetes/
/ $
/ $ vault write auth/kubernetes/config \
>     kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"
Success! Data written to: auth/kubernetes/config
/ $ vault policy write extended-api-fetcher - <<EOF
> path "internal/data/token/api-client" {
>   capabilities = ["read"]
> }
> EOF
Success! Uploaded policy: extended-api-fetcher
/ $ vault write auth/kubernetes/role/extended-api-fetcher \
>   bound_service_account_names=extended-api-fetcher \
>   bound_service_account_namespaces=default \
>   policies=extended-api-fetcher \
>   ttl=24h
Success! Data written to: auth/kubernetes/role/extended-api-fetch
```

3. Build and Push the Docker Image

Ali's Flask application reads the `API_URL` and `X-API-Token` header to authorize. Build and push:

```
# Build the image (no cache)
docker build . -t oshaheen1882051/extended-api-fetcher:v-1.0.0 --no-cache

# Push to Docker Hub
docker push oshaheen1882051/extended-api-fetcher:v-1.0.0
```



4. Prepare the Helm Chart

1. Create a new chart:

   ```
   helm create extended-api-fetcher
   ```

2. Update `values.yaml`:

   ```
   image:
     repository: oshaheen1882051/extended-api-fetcher
     pullPolicy: Always
     tag: "v-1.0.0"

   serviceAccount:
     create: true
     automount: true
     name: extended-api-fetcher

   vault:
     role: extended-api-fetcher
     secretPath: internal/data/token/api-client
     apiUrlKey: api_url
     tokenKey: token
   ```

```
service:
  type: ClusterIP
  port: 8080

livenessProbe:
  httpGet:
    path: /healthz
    port: http
  initialDelaySeconds: 10
  periodSeconds: 10

readinessProbe:
  httpGet:
    path: /healthz
    port: http
  initialDelaySeconds: 5
  periodSeconds: 5
```

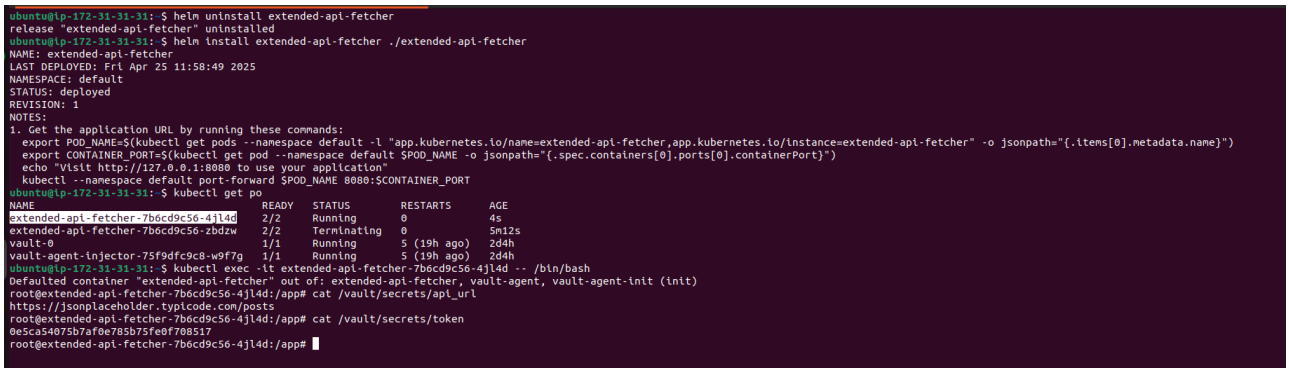
3. Add Vault annotations in `templates/deployment.yaml`:

```
metadata:
  annotations:
    vault.hashicorp.com/agent-inject: "true"
    vault.hashicorp.com/role: "{{ .Values.vault.role }}"
    vault.hashicorp.com/agent-inject-secret-api_url: "{{
.Values.vault.secretPath }}"
    vault.hashicorp.com/agent-inject-template-api_url: |
      {{`{{- with secret "`}}{{ .Values.vault.secretPath }}{{`" -}}`}}
      {{`{{ .Data.data.api_url }}`}}
      {{`{{- end }}`}}
    vault.hashicorp.com/agent-inject-secret-token: "{{
.Values.vault.secretPath }}"
    vault.hashicorp.com/agent-inject-template-token: |
      {{`{{- with secret "`}}{{ .Values.vault.secretPath }}{{`" -}}`}}
      {{`{{ .Data.data.token }}`}}
      {{`{{- end }}`}}
```

5. Deploy and Test

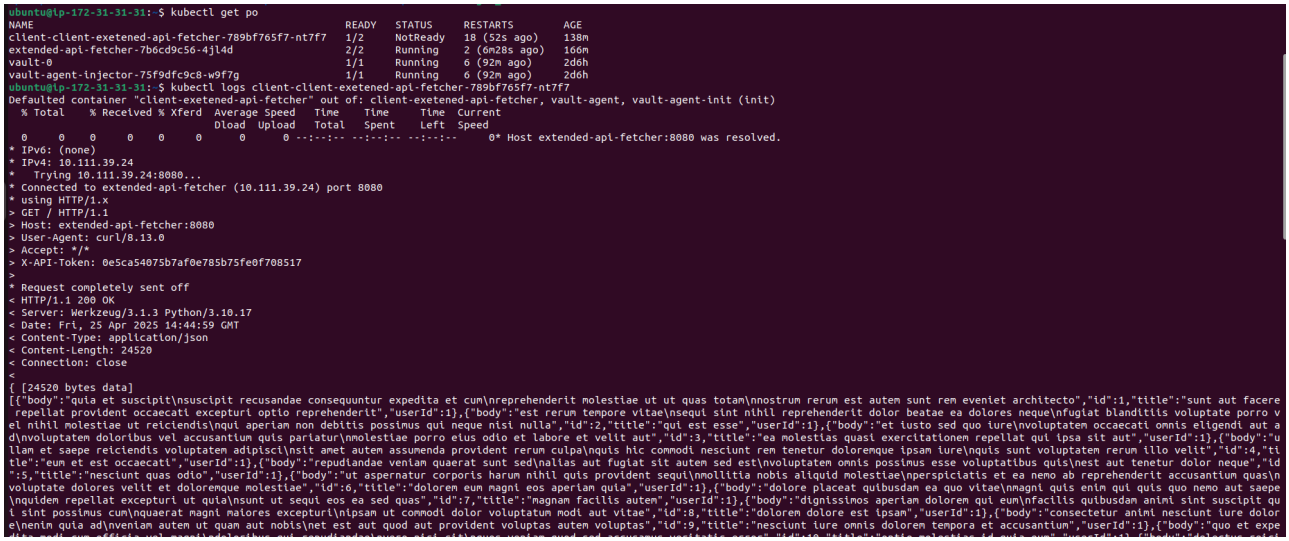
1. Install the Helm chart:

```
helm install extended-api-fetcher ./extended-api-fetcher
```



2. Port-forward and call the service with the token in a header:

```
kubectl port-forward svc/extended-api-fetcher 8080:8080

curl -v \
  -H "X-API-Token: 0e5ca54075b7af0e785b75fe0f708517" \
  http://localhost:8080/
```



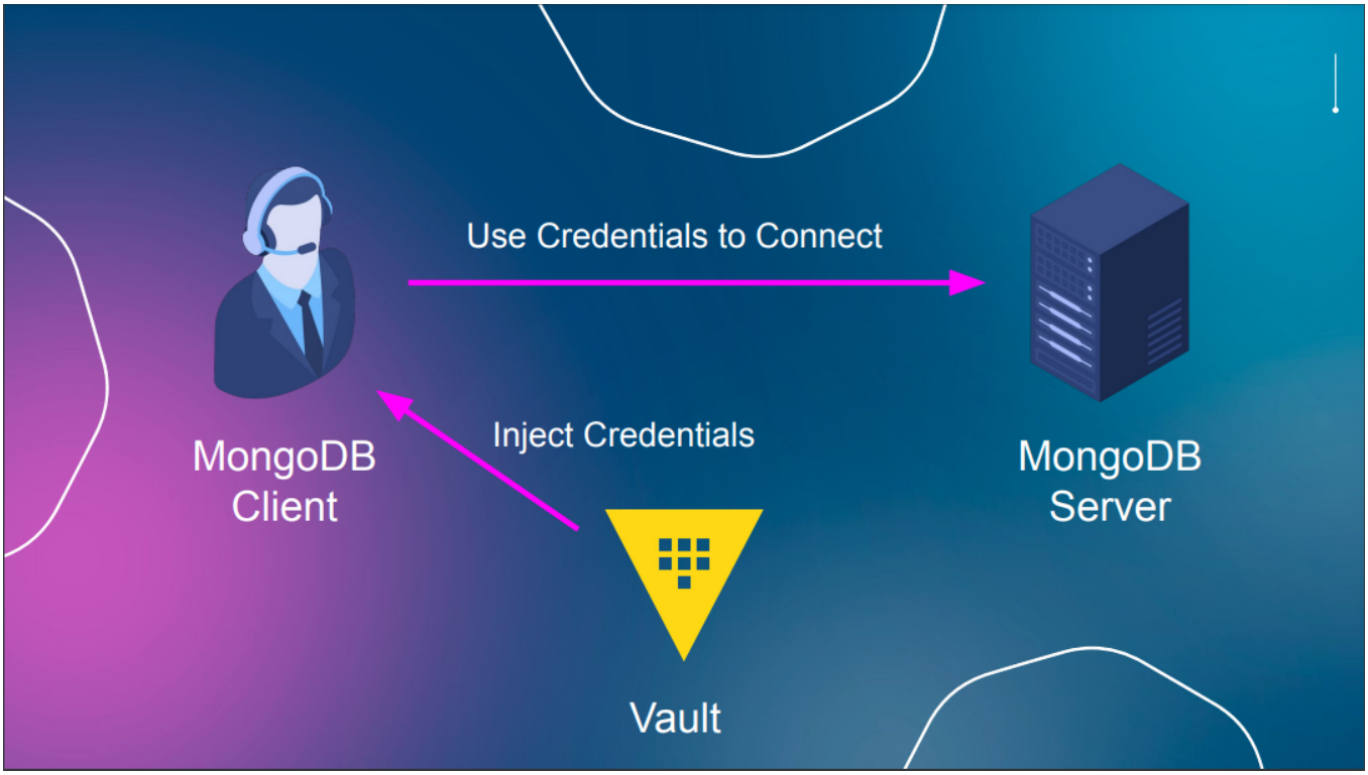
3. Check server logs to confirm authorization succeeded:

```
ubuntu@ip-172-31-31-31:~$ kubectl logs extended-api-fetcher-7b6cd9c56-4jl4d
Defaulted container "extended-api-fetcher" out of: extended-api-fetcher, vault-agent, vault-agent-init (init)
 * Serving Flask app 'main'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://10.244.0.98:8080
Press CTRL+C to quit
10.244.0.1 - - [25/Apr/2025 14:43:14] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:16] "GET /healthz HTTP/1.1" 200 -
10.244.0.96 - - [25/Apr/2025 14:43:22] "GET / HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:24] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:26] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:34] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:36] "GET /healthz HTTP/1.1" 200 -
10.244.0.96 - - [25/Apr/2025 14:43:41] "GET / HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:44] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:46] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:54] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:43:56] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:04] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:06] "GET /healthz HTTP/1.1" 200 -
10.244.0.96 - - [25/Apr/2025 14:44:11] "GET / HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:14] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:16] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:24] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:26] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:34] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:36] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:44] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:46] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:54] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:44:56] "GET /healthz HTTP/1.1" 200 -
10.244.0.96 - - [25/Apr/2025 14:44:59] "GET / HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:04] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:06] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:14] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:16] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:24] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:26] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:34] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:36] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:44] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:46] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:54] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:45:56] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:46:04] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:46:06] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:46:14] "GET /healthz HTTP/1.1" 200 -
10.244.0.1 - - [25/Apr/2025 14:46:16] "GET /healthz HTTP/1.1" 200 -
ubuntu@ip-172-31-31-31:~$
```

4. Verify the JSON data returned from the API:

Use Case 3: Database (By Yehia)

In this section, we will create `MongoDB Server` with hardcoded username/password, save them as secrets inside `vault`, and inject them into `MongoDB Client`



1. Setup MongoDB Server

A `MongoDB Server` was setup using helm chart with the following `values.yaml` file:

```
image:
  repository: mongo
  tag: latest
  pullPolicy: IfNotPresent

service:
  type: ClusterIP
  port: 27017
  nodePort: null

auth:
  rootUser: admin
  rootPassword: mypassword
  enabled: true

persistence:
  enabled: false
```

with the following hardcoded credentials:

- username: adming
- password: mypassword

```
ubuntu@ip-172-31-39-10:~/project/mongodb$ kubectl get pod
NAME                                READY   STATUS    RESTARTS      AGE
flask-login-app-77dd4ff646-nfmps    1/1     Running   3 (14h ago)   5d18h
mongodb-client                      2/2     Running   0             39m
mongodb-mongodb-6fd997dfbd-7hms6    1/1     Running   1 (14h ago)   17h
vault-0                             1/1     Running   3 (14h ago)   5d20h
vault-agent-injector-75f9dfc9c8-xdn4n   1/1     Running   3 (14h ago)   5d20h
```

2. Setup Secrets

Let's store the previous credentials into vault and verify that they got stored

```
ubuntu@ip-172-31-39-10:~/project$ kubectl exec -it vault-0 -- /bin/sh
/ $ vault kv get internal/database/config
======== Secret Path ========
internal/data/database/config

======= Metadata =======
Key                Value
---                -----
created_time       2025-04-24T12:57:31.266526796Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            1

====== Data ======
Key         Value
---         -----
password    mypassword
username    admin
```

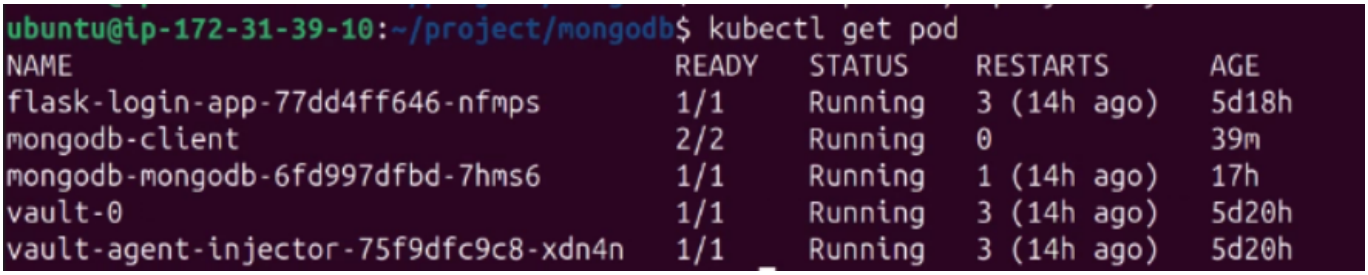
3. Setup MongoDB Client

Let's setup a `MongoDB Client Pod` using the following `mongodb-client-pod.yaml` file

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb-client
  annotations:
    vault.hashicorp.com/agent-inject: "true"
    vault.hashicorp.com/role: "internal-app"
    vault.hashicorp.com/agent-inject-secret-mongodb-creds:
"internal/data/database/config"
    vault.hashicorp.com/agent-inject-template-mongodb-creds: |
      {{- with secret "internal/data/database/config" -}}
      export MONGO_USER="{{ .Data.data.username }}"
      export MONGO_PASS="{{ .Data.data.password }}"
      {{- end }}
spec:
  serviceAccountName: internal-app
  containers:
```

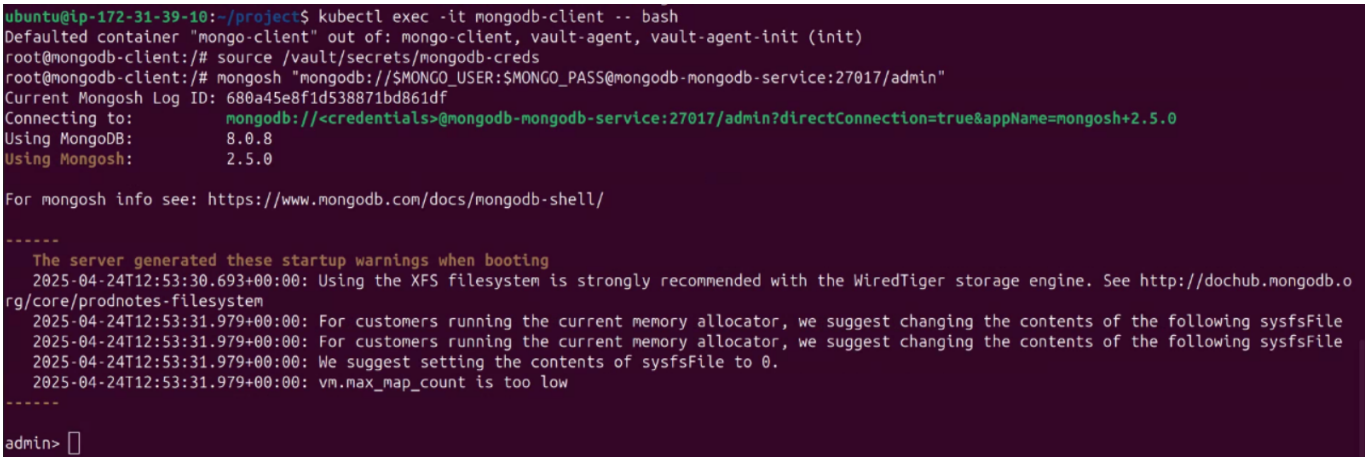
```
    - name: mongo-client
      image: mongo:6.0
      command: ["sleep", "infinity"]
```

such that our secrets will get injected into it



4. Connect to The Server

Let's access the client pod and attempt to connect to the server using the injected secrets



A connection got established successfully!

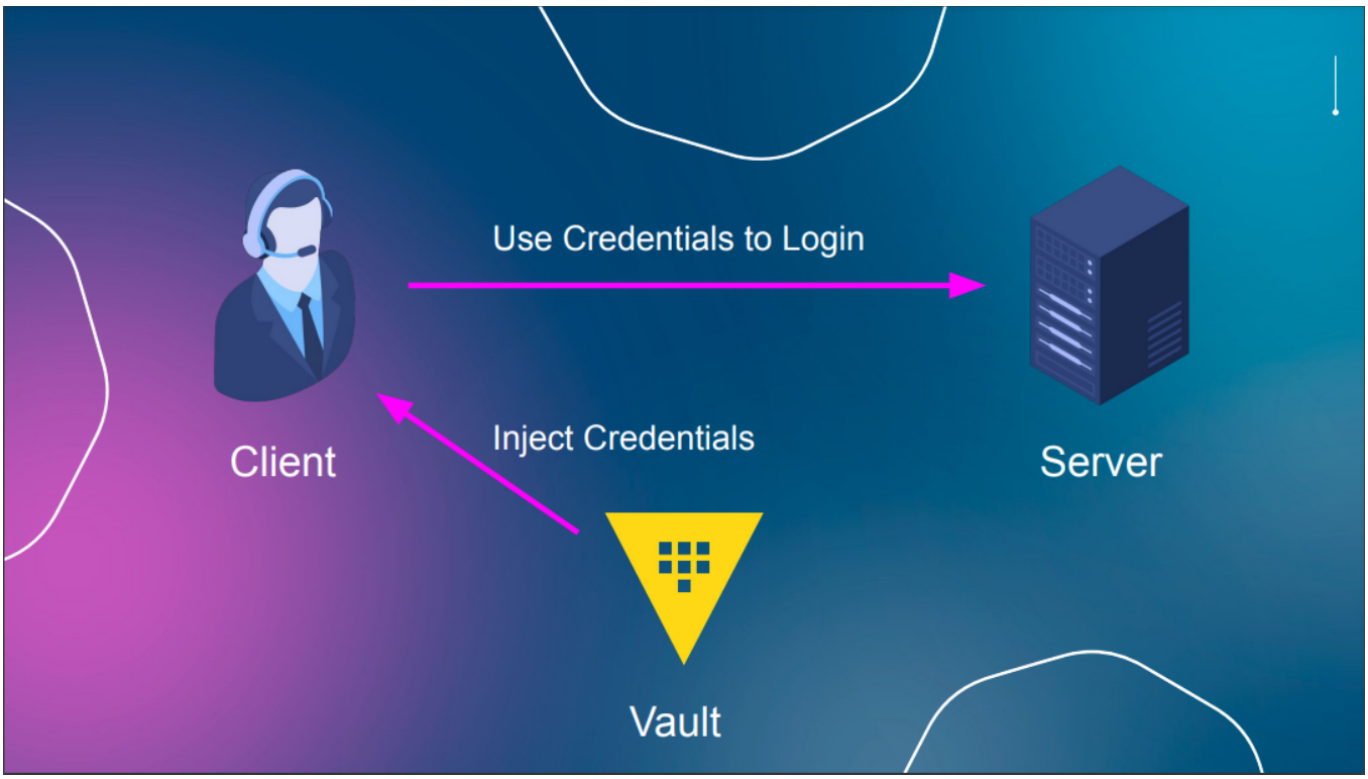
5. Testing

Let's execute few commands to check that the database functions correctly

Everything works as expected!

Use Case 4: Login System (By Ammar)

In this section we will create a `Flask-App Server` with hardcoded username/password, save them as secrets inside `vault`, and inject them to a client pod

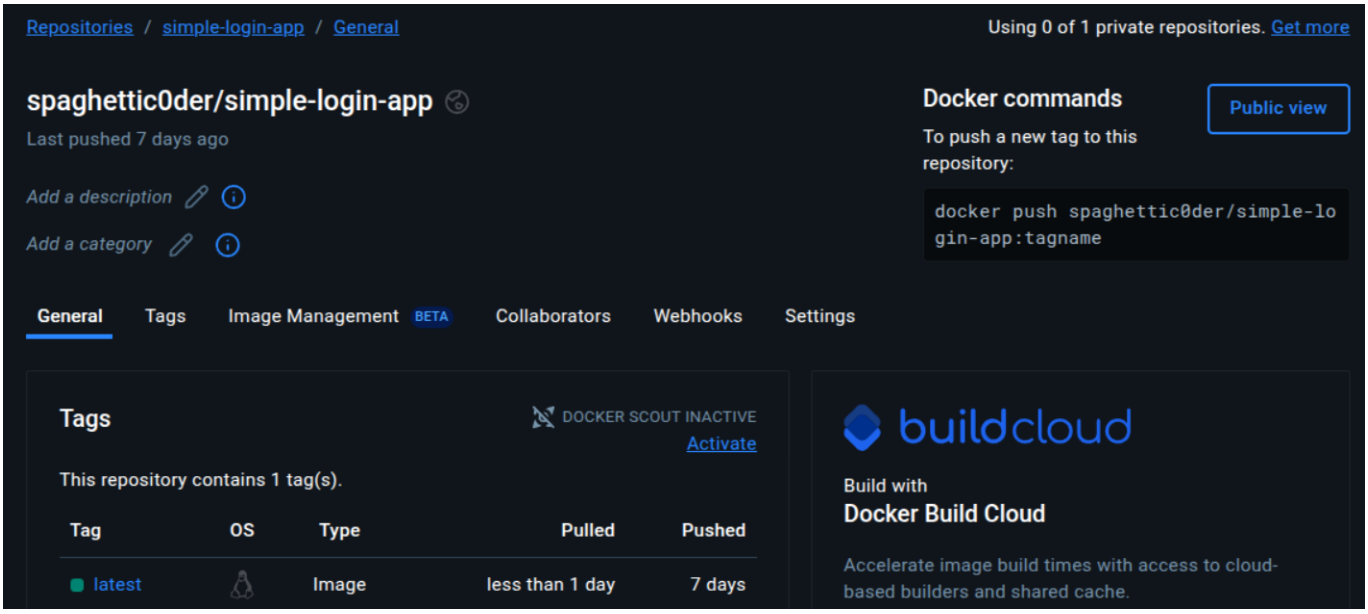


1. Create Flask-App

Let's mock up a server by creating a `Flask-App` with hardcoded credentials which accepts post request and returns a successful response only when the recieved credentials match the hardcoded ones

```
from flask import Flask, request, jsonify

app = Flask(__name__)

USERS = {
    "admin": "admin"
}

@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')

    if username in USERS and USERS[username] == password:
        return jsonify({"message": "Login successful!"}), 200
    else:
        return jsonify({"error": "Invalid credentials"}), 401

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

and then let's dockerize this app and push it to `Dockerhub`



Link for the created docker image

2. Deploy The App Using Helm

Let's deploy the pushed docker image using helm chart with the following `values.yaml` file

```
replicaCount: 1
image:
  repository: spaghettic0der/simple-login-app
  pullPolicy: IfNotPresent
  tag: "latest"

serviceAccount:
  create: true
  automount: true
  annotations: {}
  name: ""

service:
  type: ClusterIP
  port: 5000
ingress:
  enabled: false
  className: ""
  annotations: {}
  hosts:
    - host: chart-example.local
      paths:
        - path: /
          pathType: ImplementationSpecific

livenessProbe:
```

```
    httpGet:
      path: /
      port: http
  readinessProbe:
    httpGet:
      path: /
      port: http

  autoscaling:
    enabled: false
    minReplicas: 1
    maxReplicas: 100
    targetCPUUtilizationPercentage: 80
```

```
ubuntu@ip-172-31-39-10:~/project/mongodb$ kubectl get pod
NAME                                READY   STATUS    RESTARTS      AGE
flask-login-app-77dd4ff646-nfmps    1/1     Running   3 (14h ago)   5d18h
mongodb-client                      2/2     Running   0             39m
mongodb-mongodb-6fd997dfbd-7hms6    1/1     Running   1 (14h ago)   17h
vault-0                             1/1     Running   3 (14h ago)   5d20h
vault-agent-injector-75f9dfc9c8-xdn4n   1/1     Running   3 (14h ago)   5d20h
```

3. Setup The Secrets

Let's store the previous credentials into vault and verify that they got stored

```
ubuntu@ip-172-31-39-10:~/project$ kubectl exec -it vault-0 -- /bin/sh
/ $ vault kv get internal/secrets/config
======== Secret Path ========
internal/data/secrets/config

======= Metadata =======
Key                Value
---                -----
created_time       2025-04-24T15:20:43.312940351Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            1

====== Data ======
Key         Value
---         -----
password    admin
username    admin
```

4. Create Client Pod

Let's create a client pod which attempts to make a post request to the server using the following `client-pod.yaml`

```
  apiVersion: v1
  kind: Pod
  metadata:
```

```
    name: login-client
    annotations:
      vault.hashicorp.com/agent-inject: "true"
      vault.hashicorp.com/role: "internal-app"
      vault.hashicorp.com/agent-inject-secret-flask-creds:
  "internal/data/secrets/config"
      vault.hashicorp.com/agent-inject-template-flask-creds: |
        {{- with secret "internal/data/secrets/config" -}}
        username={{ .Data.data.username }}
        password={{ .Data.data.password }}
        {{- end }}
  spec:
    serviceAccountName: internal-app
    containers:
    - name: app
      image: alpine/curl:latest
      command: ["/bin/sh", "-c"]
      args:
        - |
          echo "Waiting for Vault secrets...";
          while [ ! -f /vault/secrets/flask-creds ]; do sleep 1; done;
          . /vault/secrets/flask-creds;
          echo "Attempting login with username: $username";
          curl -X POST http://flask-login-app:5000/login \
            -H "Content-Type: application/json" \
            -d "{\"username\":\"$username\", \"password\":\"$password\"}";
          sleep 3600
```

```
ubuntu@ip-172-31-39-10:~/project/mongodb$ kubectl get pod
NAME                                    READY   STATUS    RESTARTS      AGE
flask-login-app-77dd4ff646-nfmps        1/1     Running   3 (14h ago)   5d18h
mongodb-client                          2/2     Running   0             39m
mongodb-mongodb-6fd997dfbd-7hms6        1/1     Running   1 (14h ago)   17h
vault-0                                 1/1     Running   3 (14h ago)   5d20h
vault-agent-injector-75f9dfc9c8-xdn4n   1/1     Running   3 (14h ago)   5d20h
```

5. Verify

Let's check the logs to verify that the post request which was sent by the client pod got a successful response by checking the logs of the client pod

```
ubuntu@ip-172-31-39-10:~/project$ kubectl logs login-client
Defaulted container "app" out of: app, vault-agent, vault-agent-init (init)
Waiting for Vault secrets...
Attempting login with username: admin
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
{"message":"Login successful!"}
100    72  100    32  100    40   5967   7459 --:--:-- --:--:-- --:--:-- 14400
```

The logs indicate a successful response!

Presentation link

Demo link

Repo link